About cache associativity in low-cost shared memory multi-microprocessors

Nathalie Drach, Alain Gefflaut, Philippe Joubert, André Seznec

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET AUTOMATIQUE

About cache associativity in low-cost shared memory multi-microprocessors

Nathalie Drach, Alain Gefflaut, Philippe Joubert, André Seznec

N° 2083

Octobre 1993

PROGRAMME 1

Architectures parallèles,

bases de données,

réseaux et systèmes distribués

ISSN 0249-6399

About cache associativity in low-cost shared memory multi-microprocessors

Nathalie Drach, Alain Gefflaut, Philippe Joubert, André Seznec*

Programme 1 — Architectures parallèles, bases de données, réseaux et systèmes distribués
Projet Calcpar et LSP

Rapport de recherche n°2083 — Octobre 1993 — 20 pages

Abstract: In 1993, sizes of on-chip caches on current commercial microprocessors range from 16K bytes to 36 Kbytes. These microprocessors can be directly used in the design of a low cost single-bus shared memory multiprocessors without using any second-level cache.

In this paper, we explore the viability of such a multi-microprocessor. Simulations results clearly establish that performance of such a system will be quite poor if on-chip caches are direct-mapped. On the other hand, when the on-chip caches are partially associative, the achieved level of performance is quite promising.

In particular, two recently proposed innovative cache structures, the skewed associative cache organization and the semi-unified cache organization are shown to work fine.

Key-words: microprocessors, shared memory multi-microprocessors, cache, skewed-associative cache, semi-unified cache.

(Résumé : tsvp)

*e-mail : drach, gefflaut, pjoubert, seznec@irisa.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : (33) 99 84 71 00 – Télécopie : (33) 99 38 38 32

De l'associativité des caches dans les multi-microprocesseurs à coût modéré

Résumé : Aujourd'hui la taille des caches internes des microprocesseurs permet d'envisager la construction de multi-microprocesseurs à bus commun à coût extrèmement modéré, c'-à-d sans cache secondaire.

Dans ce rapport, nous étudions l'impact de l'assopciativité sur des caches internes sur les performances de tels systèmes. En particulier, les caches skewed-associatifs et semi-unifiés sont évalués.

Mots-clé : multi-microprocesseurs, caches, caches associatifs

1 Introduction

Many commercial shared memory multi-microprocessor systems are built around a single bus and a single memory as illustrated in figure 1: n processors are sharing a single bus and a single memory. Encore Multimax [1], SGI Challenge or Sparc Station 10 are examples of such multiprocessors. On such systems, the access to the bus and the memory is generally

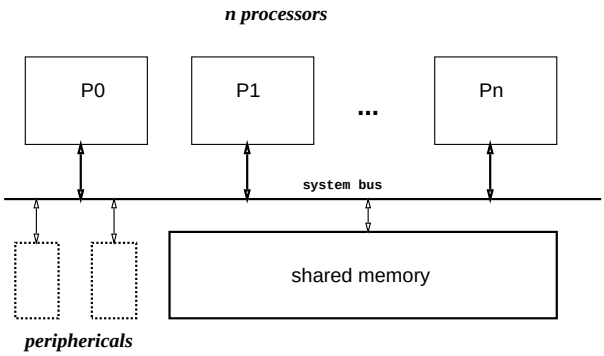


Figure 1: General structure of a single bus single shared memory multiprocessor

the bottleneck for performance: reducing the traffic on the bus and memory is a dramatic issue. In order to limit this traffic, caches are associated with the processors.

In most single bus single shared memory multi-microprocessor systems (e.g. Sparc Station 10), a second-level cache is associated with each microprocessor. The hardware cost induced by this second-level cache (cache memories, board space,..) may predominant over the cost of the microprocessor itself. On the other hand, as they include on-chip cache coherency mechanisms, most of the newly introduced second generation RISC microprocessors (e.g. MIPS R4000) may be used directly in the design of a shared memory multiprocessor without second-level caches. Then relatively a cheap single bus shared memory multi-microprocessor may be built. In such a system, the chip count would remain very limited; e.g. the overcost of an 8-processor system over a 4-processor system will essentially consist in the four added microprocessor chips.

The aim of this paper is to explore the viability of such an approach considering current available integration density. On current commercially available microprocessors, on-chip cache sizes range from 16Kbytes (e.g. MIPS R4000 [9]) to 36Kbytes (TI SuperSparc [12]). As performance of a multi-microprocessor system will essentially depend on the bus and memory traffic, we investigate using some associativity on on-chip caches and particularly the use of two recently proposed partially associative cache organizations, the skewed associative cache [14, 13] and the semi-unified cache [5].

The remainder of this paper is organized as follows. In section 2, we briefly describe the skewed-associative cache organization and its characteristics; in section 3, the semi-unified cache organization is presented. Then section 4 describes our simulation methodology.

Execution driven simulation results are given in section 5. These simulation results clearly indicate that, using direct-mapped cache structures will lead to quite low performance level, while using some associativity degree on on-chip caches will allow a higher performance level. Performance achieved when using skewed-associative caches or semi-unified caches is particularly encouraging.

2 Skewed-associative caches

The skewed-associative cache organization has been recently introduced in [14, 13].

2.1 Skewing on caches: principle

A set-associative cache is illustrated by Figure 2: a X way set-associative cache is built with X distinct banks. A line of data with base address D may be physically mapped on physical line $f(D)$ in any of the distinct banks.

In a *skewed-associative cache*, different mapping functions are used for the distinct cache banks i.e., a line of data with base address D may be mapped on physical line $f_0(D)$ in cache bank 0 or in $f_1(D)$ in cache bank 1, etc.

This very slight modification in the design is illustrated in Figure 3.

2.2 Choosing the skewing functions

Insight on the properties that might exhibit functions chosen for skewing the lines in the distinct cache banks in order to obtain a good hit ratio is given in [14, 13]. Here we only recall the inter-bank dispersion property.

In a usual X-way set-associative cache, when (X+1) lines of data contend for the same set in the cache, they are all conflicting for the same place in the X cache banks: one of the lines must be evicted from the cache (Figure 2).

Skewed-associative caches were introduced to avoid this situation by scattering the data: mapping functions can be chosen such that whenever two lines of data conflict for a single location in cache bank i, they have very low probability to conflict for a location in cache bank j (Figure 3).

Ideally, mapping functions may be chosen such as the set of lines that might be mapped on a cache line of bank i will be equaly distributed over all the lines in the other cache banks.

Nevertheless it was shown in [14] that very partial inter-bank dispersion gives equivalent performance as complete inter-bank dispersion.

2.3 A family of skewing functions

The family of skewing functions used for the simulations presented in the paper are based on manipulations on bit strings in addresses of data. The degrees of skewed associativity that were simulated are 2 and 4.

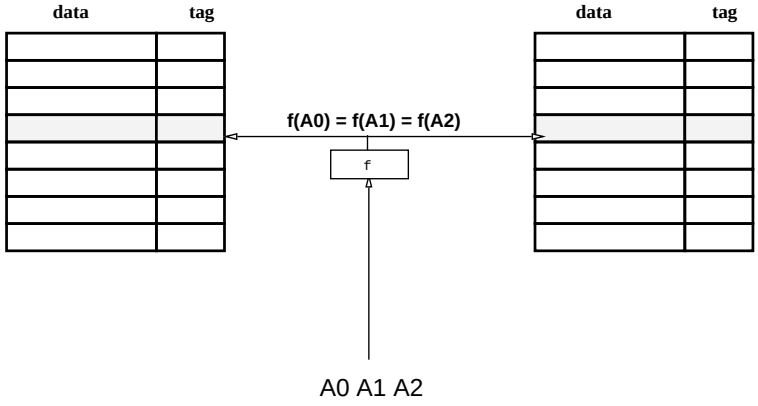


Figure 2: 3 data conflicting for a single set on a two-way set-associative cache

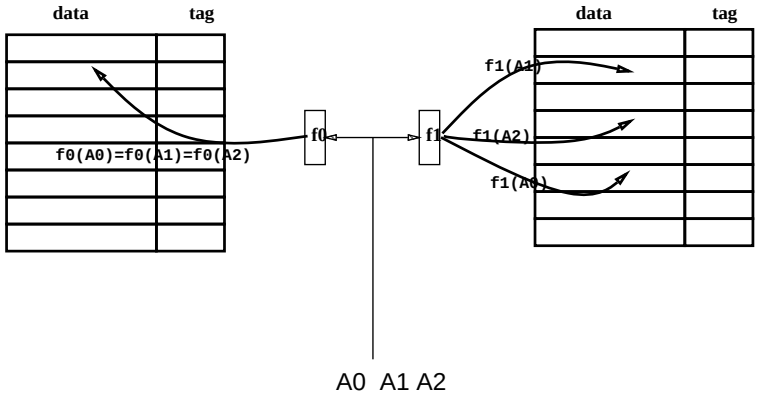


Figure 3: Data conflicting for a cache line on bank 0, but not on bank 1 on a skewed-associative cache

From now, we consider that the cache consists in X cache banks 2^{n+6} cache lines of 2^c bytes.

Let us consider the decomposition of a binary representation of an address A in bit substrings A = (A_3 ,A_2, A_1, A_0), A_0 is a $c+n$ bits string. A_1 and A_2 are two 6 bits strings and A_3 is the string of the most significant bits.

Let $(y_6, y_5, .., y_1)$ be the binary representation of $Y = \Sigma_{i=1,6} y_i 2^{i-1}$. Let us consider the function H defined as follows:

$$\begin{array}{rccl} H: & \{0,..,2^6-1\} & \longrightarrow & \{0,..,2^6-1\} \\ & (y_6, y_5, .., y_1) & \longrightarrow & (y_6 \oplus y_1, y_6, y_5, .., y_3, y_2) \end{array}$$

where $\oplus$ is the XOR (exclusive or) operation.
Let us consider the four mapping functions from the main address space S in the cache address space defined respectively by:

$$\begin{array}{rccl} f_0: & S & \longrightarrow & \{0,..,2^6-1\} * \{0,..,2^{c+n}-1\} \\ & (A_3, A_2, A_1, A_0) & \longrightarrow & (H(A_1) \oplus H^{-1}(A_2) \oplus A_2, A_0) \\ \\ f_1: & S & \longrightarrow & \{0,..,2^6-1\} * \{0,..,2^{c+n}-1\} \\ & (A_3, A_2, A_1, A_0) & \longrightarrow & (H(A_1) \oplus H^{-1}(A_2) \oplus A_1, A_0) \\ \\ f_2: & S & \longrightarrow & \{0,..,2^6-1\} * \{0,..,2^{c+n}-1\} \\ & (A_3, A_2, A_1, A_0) & \longrightarrow & (H^{-1}(A_1) \oplus H(A_2) \oplus A_2, A_0) \\ \\ f_3: & S & \longrightarrow & \{0,..,2^6-1\} * \{0,..,2^{c+n}-1\} \\ & (A_3, A_2, A_1, A_0) & \longrightarrow & (H^{-1}(A_1) \oplus H(A_2) \oplus A_1, A_0) \end{array}$$

Using these functions for mapping data on the cache ensures a reasonable dispersion of the addresses over the cache banks:

$$\forall\ D\ , \forall\ d\ \ \forall\ i \neq j\ , if\ \{f_i(D) = f_i(d)\ \&\ f_j(D) = f_j(d)\}\ then\ \ d \equiv D\ mod\ 2^{c+n+12}$$

or in less mathematical terms: the set of data that are mapped onto a single line by mapping function f_0 is splitted on 64 lines by the other mapping functions.

Hardware computations of these functions is quite simple (only a few XOR gates)

More precise justifications for this choice of skewing functions may be found in [13] and in [14].

2.4 Replacement policy

LRU replacement strategy cannot be implemented at a reasonable hardware cost on a skewed associative cache. Nevertheless, pseudo-LRU strategy with the same tag memorization cost as a LRU replacement strategy on a set-associative cache has been shown to be effective.

2.5 Hardware cost

Using the proposed skewed associative cache organization in place of a set-associative cache organization will induce a marginal hardware overcost in the microprocessor: essentially the mechanism to compute the previously described functions f_i i.e. a very few XOR gates.

3 Semi-unified caches

The major argument that is advanced for using direct-mapped caches rather than set (or skewed) associative caches is a shorter cache hit time (i.e. the delay for accessing a data in the cache on a hit) [8].

A cache read on a direct-mapped cache may be decomposed into two consecutive steps:

1. Read the word and associated tags in cache
2. Check the tags against the address of the data

While a cache read on a n-way set-associative cache consists in three consecutive steps:

1. Read a set of n words and associated tags
2. n parallel tag checks against the address of the data
3. Selection of the correct word in the set

This extra step induces a higher hit time on a set-associative cache than on a direct-mapped cache, but differences between these hit times may not be very significant [1].

Nevertheless, when using a direct-mapped cache, data (or instructions) flowing out from the cache may be directly used after step 1, because checking the validity of the data word may be executed in parallel with other pipeline activities. The current pipeline cycle is canceled if the data (or instructions) is found to be unvalid.

Using such an optimistic execution on a direct-mapped cache allows to obtain a cache hit time significantly lower than on a classical set-associative cache (15-30% are reported). [2]

The semi-unified cache organization was introduced in [5] in order to maintain a low cache hit time while introducing some on-chip cache associativity.

A semi-unified cache organization consists of two distinct instruction/data caches where the instruction cache (resp. data cache) is used as the secondary cache for data (resp. instructions).

3.1 Sequencing a request on a semi-unified cache

The sequencing of an instruction request at address A is as follows:

1. the request is presented to the I-cache :
2. on a miss, the request is presented to the D-cache. From now, this first-level cache miss is called an *on-chip miss*:

[1] 2% was reported by Hill [8]

[2] Notice that optimistic execution is also possible with a set-associative cache [4] but at a significantly higher implementation cost.

- On a hit in the data cache, the requested line is brought back in the instruction cache; a line must be rejected from the I-cache, the replacement of this line is discussed in the next section.
- On a miss in the D-cache, the request is presented to an external cache or the main memory. This second-level cache miss will be called an *off-chip miss*.

3.2 Replacement strategy for direct-mapped semi-unified cache

The semi-unified cache organization may be used with different basic cache structures for I-D caches; but this organization is particularly attractive when both caches are direct-mapped and have the same size [5].

A semi-unified cache built with two direct-mapped caches of equal sizes will be referred to as *a direct-mapped semi-unified cache*.

Off-chip misses E.g. let us consider an off-chip instruction miss:

The requested instruction line L must be stored in physical cache line $L(A)$ (where A is the instruction request address) of the I-cache the content $L1$ of line L(A) must be rejected from the I-cache.

But this rejected line $L1$ may be stored in the D-cache in physical cache line $L(A)$.

Finally the rejected line from the whole on-chip direct-mapped semi-unified cache must be chosen among a set of two lines:

1. $L1$ the content of the targeted physical line $L(A)$ in the I-cache.
2. $L2$ the content of physical line $L(A)$ in D-cache: if $L2$ is rejected, then $L1$ is moved from I-cache to D-cache.

The strategy for chosing the rejected line may be random or LRU as on a classical two-way set-associative caches.

On-chip misses Let us consider an on-chip instruction miss which hits on the secondary on-chip cache. The requested line L lies in line $L(A)$ of D-cache; this line must be brought back to line $L(A)$ in I-cache. The content L1 of line $L(A)$ in I-cache must be removed, but since line $L(A)$ in D-cache is now empty, it may be stored in this location (L1 and L2 are swapped between I-cache and D-cache): no external traffic is induced, and the global content of the on-chip cache has not changed.

3.3 Assets and drawbacks of semi-unified caches

The direct-mapped semi-unified cache organization built with two direct-mapped caches of equal sizes S has most of the advantages of both two-way set-associative cache organization and direct-mapped cache organization; it has also the assets of both unified cache organization and split cache organization:

- The cache hit time on a direct-mapped semi-unified cache is the same as when using split direct-mapped instruction/data caches of equal sizes S.

  But the off-chip miss ratio of the direct-mapped semi-unified cache is the same as that of a unified two-way set-associative cache of size 2S.
- Parallel access to instructions and data is provided; yet the on-chip cache spaces respectively devoted to instructions and data are dynamically adjusted as on a unified two-way set-associative cache.

Implementing such a semi-unified cache organization induces some additional hardware costs over the usual split cache organization: essentially, a data path is needed for swapping the lines between I-cache and D-cache. Notice that an on-chip miss induces a miss penalty even when the request hits on the secondary on-chip cache.

A performance study on the benefits of using a direct-mapped semi-unified cache in a monoprocessor system may be found in [5].

3.4 Maintaining cache coherency

For an agent external to the microprocessor, the on-chip semi-unified cache must be seen as an on-chip unified cache: in a multiprocessor, on a transaction on the system bus, both I and D caches caches on the chip must be scanned for maintaining coherency. Notice that this is already done on most microprocessors[12].

4 Simulation methodology

4.1 Simulation technique

In order to obtain accurate simulation results, we simulate address traces of real parallel applications, using an efficient execution driven simulator, based on the SPAM kernel.

SPAM [7] allows the simulation of any shared memory multiprocessor architecture on a single processor machine. SPAM implements execution driven simulation which is the only technique that gives accurate simulation results for multiprocessor architecture. It ensures that the execution providing the address trace to be simulated is exactly the one that would be observed if the application was actually executed on the simulated architecture. The basic idea of execution driven simulation is to interleave the execution and tracing of an application with the simulation of the target architecture, and to let the simulator control the execution. The main drawback of execution driven simulation is that it is difficult to implement.

SPAM provides an easy way to implement an execution-driven simulation. The SPAM kernel furnishes a trace generator based on the Abstract Execution [10], and a simulation library. The trace generator delivers address traces and a set of synchronization events to the simulator. The simulation library allows the execution of parallel applications using shared memory on a single processor machine and provides a set of primitives used to control the execution of the processes according to events generated in the address traces.

The architectural simulator is written in C and simulates each part of the architecture described in Figure 1 on a cycle by cycle basis for optimal simulation accuracy.

4.2 Benchmark Programs

Appli	Memory references	Global memory references			Shared memory references	
		% Inst	% Read	% Write	% Read	% Write
Pthor	81 M	76	6	2.16	14	1.84
Mp3d	70 M	74	4	1.4	12	8.6
Water	78 M	69	19.9	6.44	4.1	0.56
Cholesky	66 M	70	6	2.7	18	3.3

Table 1: Application characteristics

The SPAM kernel assumes a programming model where a parallel application is made of a set of processes communicating through shared memory and synchronizing with locks and barriers. We ran our simulations on a set of four parallel scientific applications from the Stanford SPLASH benchmark suite [15].

- Pthor is a logic-level circuit simulator. It uses a variant of the Chandy-Misra algorithm that avoids using a single global time and allows each element to advance its own value of time independently of the other elements. The distribution of tasks between the different application processes is realized through task queues assigned to each process. The circuit used in the simulation is composed of 5060 elements and the program runs for 5000 time steps.
- Mp3d is a rarefied fluid flow simulation program used to study the forces applied to objects. Mp3d solves a problem in three-dimensional rarefied fluid flow simulation. The algorithm simulates individual particles moving and colliding in three-dimensional space. For the simulation, we use 5000 molecules in a 14x24x7 (2646-cell) space for 80 time-steps.
- Water simulates the evolution of a system of water molecules in the liquid state. The computation is performed over a number of time-steps in a cubical box. We use 100 molecules and 2 time-steps in the simulation.
- Cholesky is a program that performs a parallel Cholesky factorization of a sparse matrix. Cholesky was run using the matrix bcsstk14 from the Boeing-Harwell benchmark matrices, and it has 1806 equations and 61648 non-zeroes.

The basic characteristics of the applications are summarized in table 1. Results are given for four processors during the parallel phase of the computation. Private memory references correspond to accesses to code and private stack or heap. Shared memory references are only data accesses in the shared memory segment. All percentage are relative to the total number of memory references.

4.3 Simulated configurations

In order to simplify simulations a single instruction issue per cycle was simulated; parallel access to data and instruction on the same cycle in the cache was considered.

The four benchmarks were simulated for 1, 4, 8 and 12 processors [3]

Cache parameters

Instruction and data caches had the same size.

Line size was 32 bytes.

Various cache structures and sizes were simulated:

- Direct-mapped
- Set-associative 2-way and 4-way
- Skewed-associative 2-way and 4-way
- Direct-mapped semi-unified

The simulated cache coherency protocol was a 5 states write invalidate protocol.

Timing considerations

On current microprocessors, main memory access time is generally around 250 ns. 100 Mhz issue rate is a current level with today's technology (e.g. MIPS R4000, MC88110); the system bus width is generally 64 bits.

The timings used in the simulations approximately match these parameters:

1. On a write on a shared line, the bus is busy for 3 cycles by the invalidation on the other caches.
2. When the missing line is supplied by an other cache, the bus is busy for 10 cycles corresponding to one address cycle, 5 cycles of cache latency and 4 successive cycles of bus occupancy by the four 64 bits words of the line.
3. On a miss request served by the memory, the bus is busy for 30 cycles corresponding to 25 cycles of memory latency and 5 cycles of bus occupancy by the address and four successive words of the line. More complex bus protocols with imbrications of requests might have been more realistic as e.g. allowing the cache of processor P2 to supply a line for processor P3 while the memory is servicing a miss for processor P1; such protocols would result in slightly higher performance than measured in our simulations [3, 6] and we are here considering a low-cost multiprocessor then the bus must not be too complex.
4. In the particular case of the semi-unified cache, a first-level cache miss does not always lead to an off-chip transaction. In our simulations, we considered that on a first-level instruction (resp. data) cache miss scanning the data (resp. instruction) cache costs a 3 cycles penalty.

[3] Due to memory size of our workstations, we were not able to simulate 12 processors for *water*

5 Simulation results

Theoretical peak performance is one Instruction Per Cycle (IPC) per processor. Figures 4 to 6 illustrate the performance achieved on the four benchmarks for various cache sizes and organizations. Figure 7 illustrates the speed-ups for a cache size of 16 Kbytes; speed-ups are measured as the ratio of the performance obtained using a particular cache organization and the performance achieved with a uniprocessor and the *same* cache organization.

Benchmarks behavior

The behavior of the four benchmarks is quite different:

- *water* has a very limited working set:
  when using a 8Kbytes skewed-associative or 4-way set-associative cache or a 2 * 8Kbytes semi-unified cache, a hit ratio superior than 0.99 is obtained leading to very high speed-up; to reach the same order of hit ratio with a 2-way set-associative cache (resp. direct-mapped cache) , 16Kbytes (resp. 32Kbytes) are necessary.
- On *pthor*, due to a large number of synchronizations, the speed-up is relatively low and it seems that there is low benefit in increasing the number of processors from 4 to 8 or 12. Nevertheless, using some associativity on the cache clearly dramatically improves performance.
- On *cholesky*, there is a quite regular benefit in increasing the size of the caches. For a monoprocessor, the performance obtained with a direct-mapped cache is close to performance obtained with associative caches, but the performance gap dramatically increases with the number of processors.

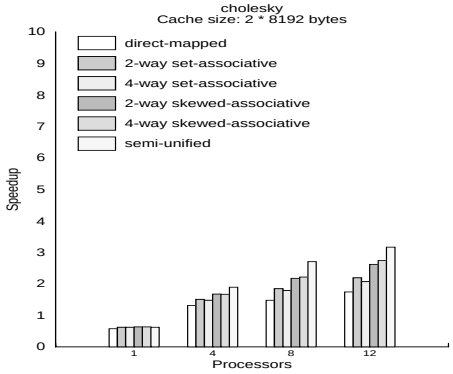
  Notice that the influence of cache size in this application is clearly illustrated by the behavior of the semi-unified cache which exhibits higher performance than the other configurations for 8Kbytes.
- On *mp3d*, increasing the cache size also results in a clear performance benefit. Particularly for 32Kbytes caches, increasing the processor number is worthwhile when some associativity is available on the cache and particularly skewed associativity.

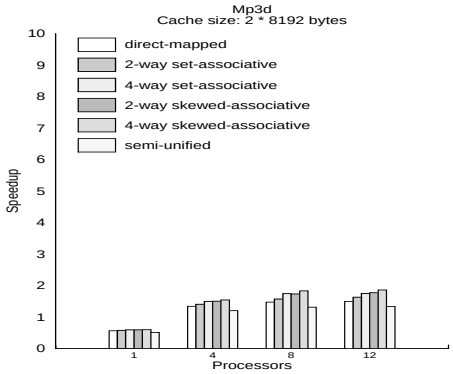
Speed-ups

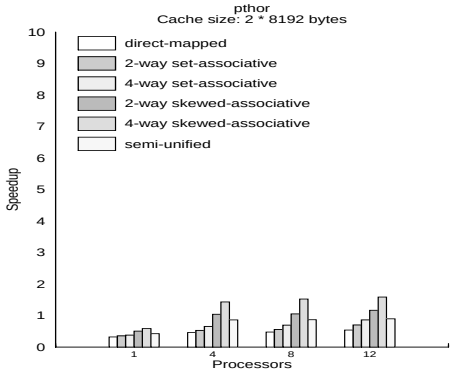
As foreseeable, using some associativity in caches improves overall IPC performance for monoprocessors, it also achieves better speed-ups (figure 7).

cholesky and *water* exhibits good speed-ups, while performance on *pthor* is not significantly increased when using 8 or 12 processors rather than 4 processors. *mp3d* would need quite large cache (32K bytes) and a good associativity mechanism to obtain a large speed-up.

Nevertheless, it seems that using 8 or 12 processors would be a good performance/price trade-off: as pointed out in the introduction, the hardware cost of an 8 or 12 processors system will be very close to the hardware cost of a 4-processor system.







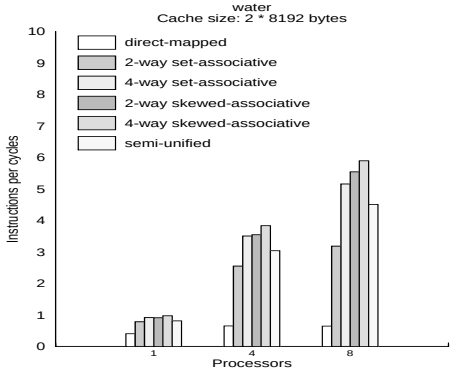
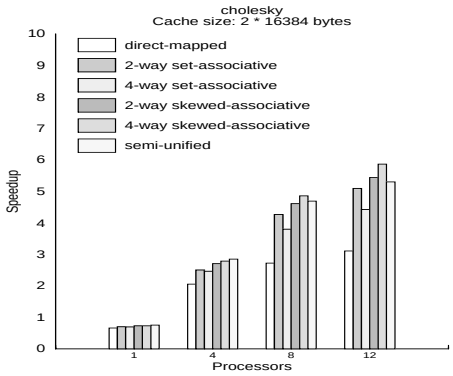
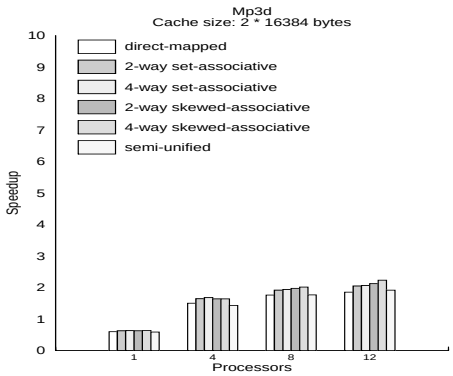
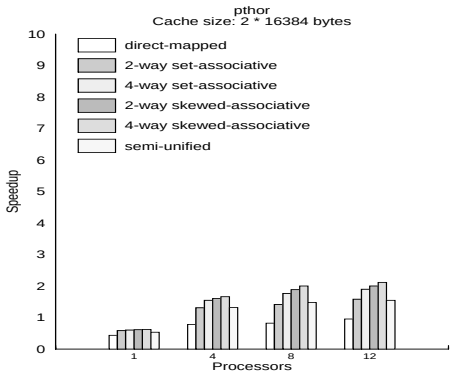


Figure 4: Performance: cache size 2 * 8 Kbytes







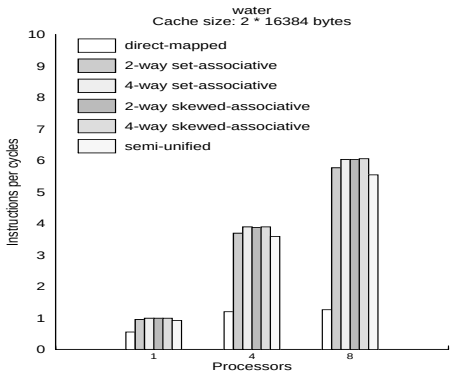
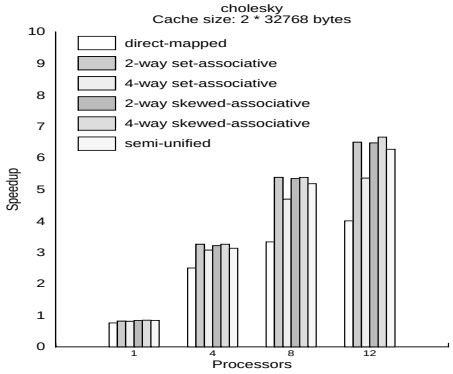
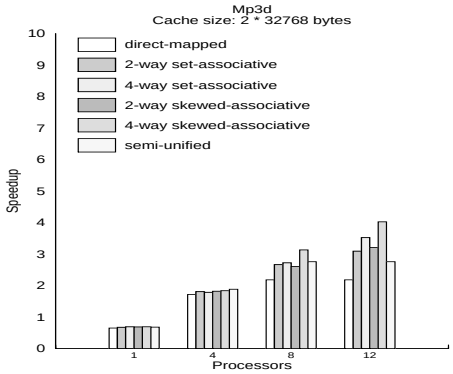
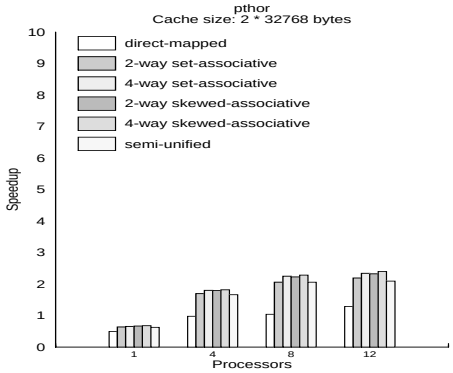


Figure 5: Performance: cache size 2* 16Kbytes







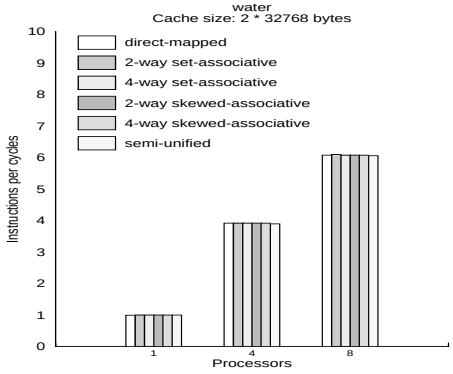
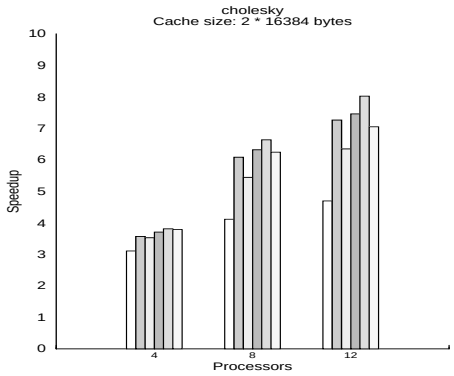
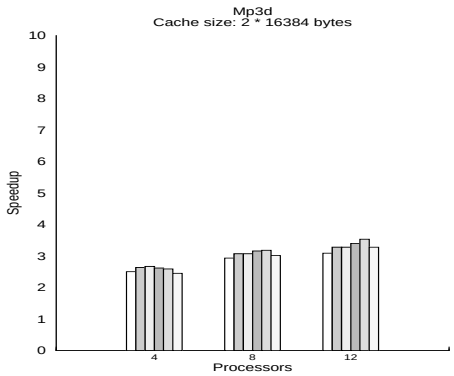
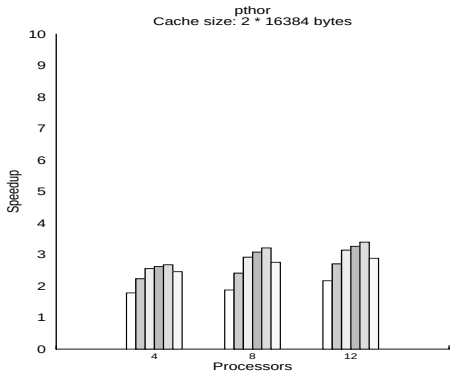


Figure 6: Performance: cache size 2* 32 Kbytes







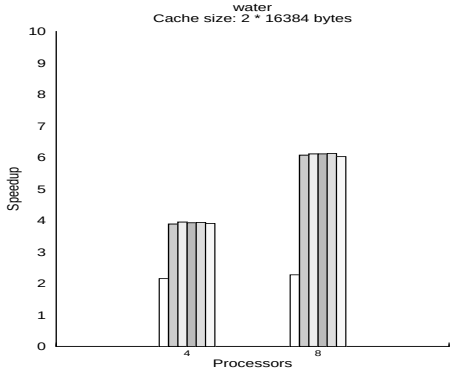


Figure 7: speedups; cache size 2* 16Kbytes

Analyzing associativity

In order to have a better evaluation of the impact of using associativity on caches in multiprocessors, we ran the simulations considering processors using 2* 256 Kbytes direct-mapped caches, then we used these performance results in order to normalize the previously reported results. We define the normalized performance of the application as the ratio $\frac{IPC\ achieved}{IPC\ achieved\ with\ same\ number\ of\ processors\ and\ 2*256K\ direct-mapped\ caches}$ Figure 8 illustrates this normalized performance for a cache size of 16 Kbytes.

Except for *mp3d* which seems to require at least a 32 Kbytes data cache, using a 2*16 Kbytes associative cache structure allows to achieve the same level of performance as a 256 Kbytes direct-mapped cache.

It may be noticed that this indicates that, when an associative cache structure is used on the microprocessor chip, a second-level direct-mapped cache smaller than 512 Kbytes will probably not enhance performance: when a second-level cache is used, the global miss penalty on a second-level cache miss is higher than the miss penalty without second-level cache, moreover maintaining cache coherency is more complex and maintaining inclusion property induces extra first level misses [2].

Direct-mapped caches

When using 2*16Kbytes direct-mapped caches, for the four applications the normalized performance decreases quite significantly when the number of processors increases: the number of cache misses is significantly higher than when using 2*256 Kbytes caches and the number of cache misses served by the caches of the other processors is more limited, then the memory is rapidly saturated.

Semi-unified caches

It must be pointed out that the performance achieved when using direct-mapped semi-unified cache organization is closer to the behavior of a set or a skewed-associative cache structure than to the behavior of a direct-mapped cache structure. This is particularly interesting because the cache hit time on a semi-unified cache is very close to the cache hit time on a direct-mapped cache.

When the cache size increases, the performance when using a semi-unified cache organization becomes slightly lower than when using set or skewed-associative split I-D cache organization; this is due to "less" associativity and on-chip cache miss penalties.

Skewed-associative caches

Performance achieved when using skewed-associative caches is higher than when using usual set-associative caches. For example, for a 2*16Kbytes cache size and a 8-processor system, the average normalized performance achieved on our set of benchmarks is 1.05 with a 4-way skewed-associative cache, 1.00 with a 2-way skewed-associative cache, 0.93 with a 4-way set-associative cache, 0.90 with a 2-way set-associative cache, 0.90 with a semi-unified cache and 0.49 with a direct-mapped cache.

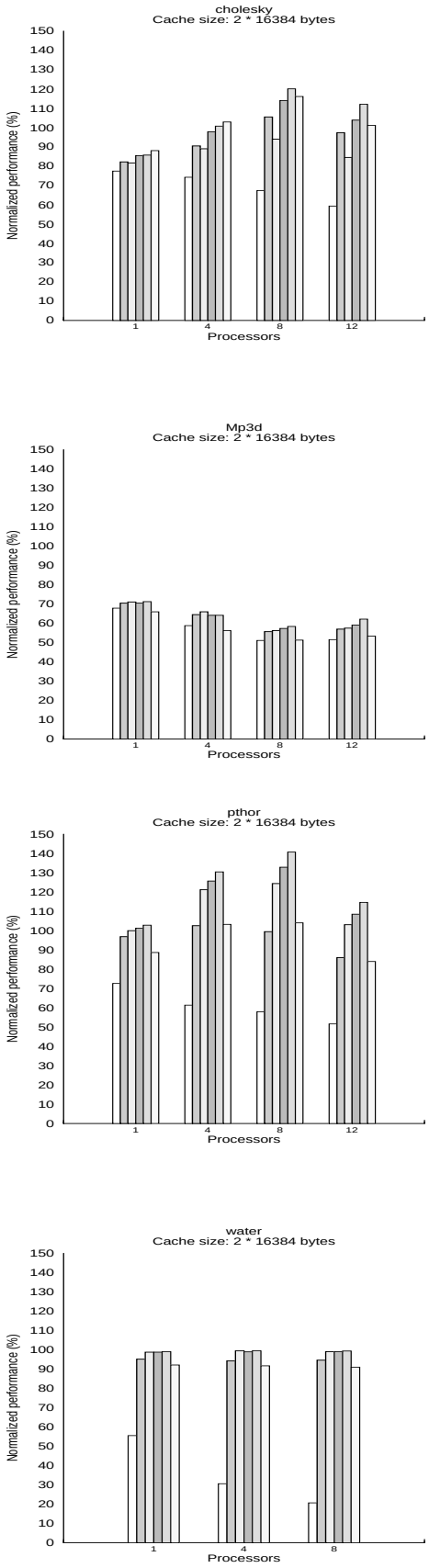


Figure 8: Normalized performance on the four benchmarks

6 Conclusion

Building a relatively low-cost shared memory multi-microprocessor system has become feasible with current microprocessor technology.

In this paper, we have explored the viability of building a single-bus single memory multi-microprocessor system without using any second-level cache. Such an approach clearly reduces the hardware cost of the system. Such a low-cost multiprocessor system may be used as a stand-alone machine. This structure may also be considered for basic clusters for non-uniform memory access (NUMA) shared memory systems [11].

Simulations were run on a scientific set of benchmarks for various cache sizes and organizations. These simulations clearly indicate that with the current level of integration technology and the current level of microprocessor performance, the speed-ups that could be obtained on such a multi-microprocessor system is highly dependant on the cache organization and mainly on its associativity. Direct-mapped cache structures will lead to very poor performance compared with using partially associative cache structures; this poor performance is due to lower monoprocessor performance conjugated with lower speed-up.

On the other hand, on our benchmark set, using a 2*16 Kbytes [4] partially associative cache organization leads to performance comparable with a 2*256Kbytes direct-mapped cache organization.

Among the partially associative caches, we have particularly focussed our attention on two innovative partially associative cache structures: the skewed-associative cache and the semi-unified cache organizations. The skewed-associative cache organization exhibits the best performance. As a skewed-associative cache has the same integration complexity as a usual set-associative cache, microprocessors designers should consider this new structure of cache.

The semi-unified direct-mapped cache organization definitely exhibits a behavior close to a set-associative cache organization while it achieves the same cache hit time as a direct-mapped cache. Clock frequency is generally determined by cache hit time; clock frequency determines peak theoretical performance. Using a semi-unified direct-mapped cache organization might be the best trade-off for industrials in order to announce high peak performance and to achieve the best level of *effective* performance.

References

[1] "Using the Encore Multimax", Tech. Mem. No 65, Rev. 1, Argonne National Laboratory, Feb. 1987

[2] Baer J.L., W.H. Wang "On the inclusion property for multi-level cache hierarchies" , Proceedings of the 15th International Symposium on Computer Architecture (IEEE-ACM), June 1988

[4] i.e. the current level of integration density

[3] M. Cekleov et al "SPARCcenter 2000: Multiprocessing in the 90's!", Proceedings Compcon Spring '93, Feb. 93

[4] J.H. Chang, H. Chao, and K. So "Cache Design of A Sub-Micro CMOS System/370" pp208-213, Proceedings of the 14th International Symposium on Computer Architecture (IEEE-ACM), May 1987.

[5] N. Drach, A. Seznec "Semi-unified Caches" Proceedings of the International Conference on Parallel Processing, August 1993

[6] J.M Frailong et al, "The Next-Generation SPARC Multiprocessing System Architecture", Proceedings Compcon Spring '93, Feb. 93

[7] A. Gefflaut, P. Joubert "*SPAM* : A Multiprocessor Execution Driven Simulation Kernel", INRIA Report, March 1993

[8] M.D. Hill, "A case for direct-mapped caches", IEEE Computer, Dec 1988

[9] G. Kane, J. Heinrich *MIPS RISC Architecture* Prentice-Hall, 1992

[10] J. Larus, "Abstract Execution : A Technique for Efficiently Tracing Programs", Software Practice and Experience, Dec 1990

[11] D. Lenoski et al, "The Stanford DASH multiprocessor", Computer, March 1992

[12] "TMS390Z50, Data Sheet", Texas Instrument, 1992

[13] A. Seznec, F. Bodin "Skewed Associative Caches" Proceedings of PARLE' 93 (Lecture Notes in Computer Science) June 1993

[14] A. Seznec, "A Case for Two-way Skewed Associative Caches", Proceedings of the 20^{th} International Symposium on Computer Architecture, May 1993

[15] J.P. Singh, W. Weber, A. Gupta "SPLASH : Stanford Parallel Applications for Shared-Memory", Technical Report CSL-TR-91-469, Stanford University, 1991.

[16] M. Slater, "Corollary Unveils 486 Multiprocessor Cache Chips", Microprocessor report, aug 1991

Unité de recherche INRIA Lorraine, Technôpole de Nancy-Brabois, Campus scientifique,
615 rue de Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399